Optimal Paging Under a Generalized Cost Function

Arvind

and

Wilfred Plouffe, Jr.

Technical Report #65

May 1975

Department of Information and Computer Science
University of California, Irvine

ABSTRACT

A generalized cost function is presented which is useful for comparing the performance of memory paging algorithms. This function is a close approximation to the real space-time produce and is expressed in terms of the number of page faults and the amount of memory occupied at the time of the fault. By using this function, it is also easy to determine the dynamic memory requirements of a program.

A demand paging algorithm is developed and shown to be optimal with respect to the cost function. Even though the algorithm is unrealizable, it is useful as a theoretical lower bound on the cost for processing any reference string.

1. Introduction

A number of schemes for automatic management of the memory space of computers have been proposed and implemented over the last 15 years. These schemes can be compared on the basis of many factors. Following is a list of factors most commonly considered.

(i) The amount of information that is assumed about a program's behavior varies from one algorithm to another. On one hand, some algorithms assume perfect knowledge of the future resulting in the so-called unrealizable algorithms. On the other extreme are the algorithms that use only the past history of the current execution of a program. Only algorithms of this type are realizable for implementation. To analyze these algorithms, various probabilistic models of program behavior are assumed [A1,A2,D2,D3,K1,S1].

(ii) An important characteristic is whether the algorithm works under a fixed partition of memory or dynamically varies the partition size.

(iii) Some times algorithms are applied globally to all the programs in the memory as opposed to the memory space of one program at a time. This

is usually to reduce the cost of implementation.

(iv) Another factor is the cost of implementing an algorithm and the cost of processing a given reference string using the algorithm. Analytic comparison of memory management algorithms always ignores the initial cost of implementation. We will concern ourselves mainly with the memory related costs incurred to the system in executing a program.

There is general agreement in the literature that if a page is to be replaced, it should be the page least likely to be referenced at that time [B1,C2,D2,M1]. However, no consensus exists on determining the amount of memory to be allocated to a program. Since the amount of information needed for efficient execution varies in time (i.e., the working set), variable partition algorithms are more appealing than those using fixed partitioning. Part of the disagreement is due to the fact that no single cost function is used to evaluate all algorithms. A product of the memory occupied and the real time spent in the system is considered a good measure, but generally is very hard to evaluate analytically. In section 2, we propose a cost function which is a very close approximation to the real space-time product and is expressed using page faults and the size of

the memory occupied at the time of the page fault. With this cost function, it becomes meaningful to ask if a page should be kept in the memory or not. We propose an algorithm in section 4 which is optimal with respect to this cost function. For the sake of generality, we have not assumed any particular model for program behavior.

2. The Cost Function

This section presents the cost function and its justification. We assume that a program's name space N is a set of n contiguous pages and that the system's (main) memory space M is a set of m contiguous page frames. We also assume that a program's paging behavior for a given set of inputs is described by its (page) reference string ω which is a sequence

$$\omega = r_1 r_2 \cdots r_T, \qquad r_t \in N \qquad (1 \leqslant t \leqslant T)$$

where $r_t = i$ means that page i is referenced at the t^{th} memory reference. Thus, there is an instance in ω for each memory reference. We also use ω_t to mean the sequence $r_t r_{t+1} \cdots r_T$. The discrete time parameter t $(1 \leqslant t \leqslant T)$ represents instants in execution or virtual time.

A paging algorithm A is an algorithm for processing a reference string ω and generating, in response, a sequence of memory states $S_0 S_1 \cdots S_T S_{T+1}$ where S_0 is a given intial

memory state and $S_{T+1}=\phi$. Each S_t is a set of pages of N which reside in M at time t and satisfies the following three conditions, for $1 \leqslant t \leqslant T$,

(1) $S_t \subseteq N$,

(2) $|S_t| \leqslant m$,

(3) $r_t \in S_t$.

Also, S_t is related to S_{t-1} by the relation

$$S_t = S_{t-1} + X_t - Y_t \tag{2.1}$$

where $X_t \subseteq N - S_{t-1}$ is the set of pages fetched from the secondary memory and $Y_t \subseteq S_{t-1}$ is the set of pages to be replaced. A paging algorithm A is a <u>demand paging algorithm</u> if $S_0 = \phi$ and

$$S_t = \begin{cases} S_{t-1} - Y_t & \text{if } r_t \in S_{t-1} \\ S_{t-1} + \{r_t\} - Y_t & \text{if } r_t \notin S_{t-1} \end{cases} \tag{2.2}$$

The choice of pages for Y_t is the prerogative of algorithm A within the constraints stated above. Finally, the pages in Y_t may be removed from memory as soon as r_t has been generated.

With this background, we may now define various costs incurred in processing a reference string. The cost function should include the following:

(i) a factor for keeping one page of program or data in main memory for one time unit while the central processor is executing our program's instructions (μ = \$/page/reference). This is a

factor for part of the space-time product.

(ii) a factor for keeping one page of information in main memory for one time unit while waiting for a missing page or waiting to be scheduled (μ_f = \$/page/reference). If we let Δ_A denote the average time, in memory cycles, that it takes to transfer a page between secondary storage and main memory and to be rescheduled, then we may define $\Delta = \mu_f \cdot \Delta_A$ (Δ = \$/page/page fault). This is a second factor of the space-time product.

(iii) a factor for using the central processor for one time unit (ρ = \$/reference).

(iv) a factor for using the channel to transfer one page between main memory and secondary storage (τ = \$/transfer). This factor is generally non-zero to account for both channel usage and the stealing of memory cycles from the CPU. This factor is also constant once the page size has been fixed. It is easy to see that our assumption implies that there is no cost differential between a request to transfer two pages of information and two requests at different time instants to transfer a single page of information.

(v) and a factor for using the central processing

unit to execute the system's page fault handler routine and to save and restore a program's state information (σ = \$/fault).

Then, let $C(A,\omega,m)$ denote the cost for processing reference string ω in a memory of size m using algorithm A. We define $C(A,\omega,m)$ to be

$$C(A,\omega,m) = \sum_{t=1}^{T}(\rho + |S_t|\cdot\mu + |Y_t|\cdot\tau) + \sum_{t=1}^{T}|X_t|\cdot(\sigma + \tau + |S_t|\cdot\Delta) + |Y_{T+1}|\cdot\tau \quad (2.3)$$

This definition may be easily simplified. First, define $PF(A,\omega,m)$ to be the set of all virtual time instants at which a page fault occurs, or

$$PF(A,\omega,m) = \{t \mid r_t \notin S_{t-1}\} \quad (2.4)$$

We will assume that each page must initially be brought into main memory to satisfy a page fault and must be removed from main memory to secondary storage before time T+1. Then, for every $t \in PF(A,\omega,m)$, there is a corresponding $Y_{t'}$ $(t+1 \leq t' \leq T+1)$ where $r_t \in Y_{t'}$ is removed from main memory; and for any $r_t \in Y_t$, there is a corresponding $t' \in PF(A,\omega,m)$ where r_t was brought into the main memory. Therefore, we may rewrite $C(A,\omega,m)$ as

$$C(A,\omega,m) = T\cdot\rho + \sum_{t=1}^{T}|S_t|\cdot\mu + \sum_{t\in PF(A,\omega,m)}(2\tau + \sigma + |S_t|\cdot\Delta) \quad (2.5)$$

or let $\gamma = 2\tau + \sigma$, then

$$C(A,\omega,m) = T\cdot\rho + \sum_{t=1}^{T} |S_t|\cdot\mu + \sum_{t\in PF(A,\omega,m)} (\gamma + |S_t|\cdot\Delta) \qquad (2.6)$$

This cost function is more general than any of the cost functions mentioned earlier. Indeed, most of those functions are special cases of $C(A,\omega,m)$. However, in spite of its generality, $C(A,\omega,m)$ still has some implicit assumptions that are discussed below.

The first assumption is that the scheduling algorithm is nonpreemptive, that is the central processing unit is switched between programs only when a page fault occurs. One is tempted to say that the preemptions can be treated similar to page faults using different values for γ and Δ. The main problem with that is preemption and waiting for a missing page might overlap, and this overlap can not be handled by $C(A,\omega,m)$. It is this same problem that prevents $C(A,\omega,m)$ from handling prefetching.

The second assumption concerns the time unit that was chosen. We recognize that the memory reference time is not constant for each reference due to such factors as memory interleaving, overlapping of instruction execution and fetching of data, nonuniform execution time of instructions, and memory contention. However, variations in memory reference times are likely to be independent of the paging

behavior. Therefore we are justified in using the average reference time. Besides, this time unit is easily understood and agrees with most conventions [B1,B2,C1,D1,P1].

The final assumption is the most serious. There are no explicit representations for costs due to such factors as channel contention and queueing delays. However, averages for these factors may be included in other parameters of $C(A,\omega,m)$ as μ, μ_f, τ, and Δ. While this may not be a completely satisfactory answer for some situations, it is sufficient for our purposes.

Thus, we have developed a generalized cost function that accounts for most of the measurable cost factors due to processing a reference string. In addition, the cost function is flexible enough to account for some factors which we can not accurately measure, but can approximate.

3. Demand Paging vs. Prepaging

Strategies for bringing information into main memory may be broadly classified into either demand or prepaging policies. A common attitude towards the prepaging, or nondemand paging, policies may be summarized as follows:

> "It can be shown that, for any nondemand paging algorithm A, one may construct a demand paging algorithm A' that produces no more faults than A on every reference string. We are therefore

> justified in restricting attention to demand paging algorithms." [D2, page 178]

The above statement and also Theorem 6.1 in Coffman and Denning [C1, page 248] are justified if the only component of the cost function is the number of page faults. However, the true cost to the system also includes factors for saving and restoring a program's state information and keeping pages in main memory while satisfying a page fault. With prepaging, the first of these two factors can be almost completely eliminated. Also, since it is possible to overlap processing and fetching through the use of prepaging, it is possible to significantly reduce the wait time for satisfying a page fault and thus reduce this factor of the cost. Thus, the true cost to the system may be reduced through the use of prepaging, even though the number of page faults remains constant. However, we will concern ourselves only with demand paging algorithms for the remainder of this paper.

4. The Algorithm

This section defines a demand paging algorithm and proves that it is optimal. We say that a paging algorithm is optimal if, for any given reference string ω, it minimizes the cost function presented in section 2. Before defining the algorithm, we first prove the following theorem

which characterizes any optimal paging algorithm.

Theorem 1: If a demand paging algorithm is optimal and $\mu \neq 0$, then either $Y_t = \phi$ or $Y = \{r_{t-1}\}$ for $1 \leqslant t \leqslant T+1$.

This theorem states that if an optimal algorithm removes a page from the main memory it does so immediately following the most recent reference to the page.

Proof: The proof is by contradiction. We will show that an algorithm A can not be optimal if there exists a k, $1 \leqslant k \leqslant T+1$, such that according to A, $Y_k^A \neq \phi$ and $Y_k^A \neq \{r_{k-1}\}$. Assume the contrary, that is A is optimal and such a k exists. Then either $r_{k-1} \in Y_k^A$ or $r_{k-1} \notin Y_k^A$.

Case 1: Assume $r_{k-1} \in Y_k^A$. Define a new algorithm B in which all the decisions (i.e. Y_t) except Y_{k-1} and Y_k are the same as A. Y_{k-1}^B and Y_k^B are defined as follows.

$$Y_{k-1}^B = Y_{k-1}^A + Y_k^A - \{r_{k-1}\}$$

$$Y_k^B = \{r_{k-1}\}$$

Now using (2.6) and observing that all the memory states except S_{k-1} are the same for both the algorithms, we can write

$$C(A,\omega,m) - C(B,\omega,m) = |S_{k-1}^A - S_{k-1}^B| \cdot \mu + |X_{k-1}| \cdot |S_{k-1}^A - S_{k-1}^B| \cdot \Delta$$

$$= |Y_k^A - \{r_{k-1}\}| \cdot [\mu + |X_{k-1}| \cdot \Delta]$$
$$> 0 \quad (\text{since } \mu \neq 0 \text{ and } Y_k^A \neq \{r_{k-1}\})$$

Therefore A can not be optimal.

Case 2: Assume $r_{k-1} \notin Y_k^A$. Again define B as in Case 1 except for

$$Y_{k-1}^B = Y_{k-1}^A + Y_k^A$$
$$Y_k^B = \phi$$

The remainder of the proof for this case is similar to that of Case 1.

Q.E.D.

This theorem characterizes the choices for Y_t that must be considered for the optimal algorithm. We can also use this information to calculate the minimal cost. Thus we recursively define $P_c(S,\omega,m)$ in such a manner that it is the minimal cost for processing ω in a memory of size m when starting with an initial memory state of $S=\phi$.

(4.1)

$$P_c(S_i,\omega_i,m) = \begin{cases} \rho + |S_i| \cdot \mu + \min\begin{Bmatrix} P_c(S_i - \{r_i\}, \omega_{i+1}, m) \\ P_c(S_i, \omega_{i+1}, m) \end{Bmatrix} & \text{if } r_i \in S_i \\ \tau + (|S_i|+1) \cdot \Delta + \rho + (|S_i|+1) \cdot \mu + \min\begin{Bmatrix} P_c(S_i, \omega_{i+1}, m) \\ P_c(S_i + \{r_i\}, \omega_{i+1}, m) \end{Bmatrix} & \text{if } r_i \notin S_i \wedge |S_i| < m \\ \infty & \text{if } r_i \notin S_i \wedge |S_i| = m \\ |S_i| \cdot \mu & \text{if } \omega_i = \lambda \text{ (empty)} \end{cases}$$

The separate components of $P_c(S,\omega,m)$ correspond to: a

reference to a page resident in memory, a reference to a page not resident in memory and memory is not full, a reference to a page not resident in memory and memory is fully utilized, and the condition when there are no more memory references.

Now, if we define an algorithm which makes the same decisions as $P_c(\phi,\omega,m)$, we would have an optimal paging algorithm. This is the motivation for defining $P_0(\omega,m)$ as follows:

$P_0(\omega,m)$: $S_0 = \phi$

$$S_t = \begin{cases} S_{t-1} & \text{if } r_t \in S_{t-1} \wedge P_c(S_{t-1}-\{r_{t-1}\},\omega_t,m) \geq P_c(S_{t-1},\omega_t,m) \\ S_{t-1}-\{r_{t-1}\} & \text{if } r_t \in S_{t-1} \wedge P_c(S_{t-1}-\{r_{t-1}\},\omega_t,m) < P_c(S_{t-1},\omega_t,m) \\ S_{t-1}+\{r_t\} & \text{if } r_t \notin S_{t-1} \wedge P_c(S_{t-1}-\{r_{t-1}\},\omega_t,m) \geq P_c(S_{t-1},\omega_t,m) \\ S_{t-1}-\{r_{t-1}\}+\{r_t\} & \text{if } r_t \notin S_{t-1} \wedge P_c(S_{t-1}-\{r_{t-1}\},\omega_t,m) < P_c(S_{t-1},\omega_t,m) \end{cases} \quad (4.2)$$

Theorem 2: $P_0(\omega,m)$ is an optimal demand paging algorithm.

Proof: It is clear that $P_0(\omega,m)$ is a demand paging algorithm and, by "the principle of optimality," is optimal.*

Q.E.D.

---

*"The principle of optimality" may be stated as follows: "An

optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." [B3, page 83]

---

5. Properties of $P_c(S,\omega,m)$

Like Belady's MIN algorithm [B1], the $P_0(\omega,m)$ algorithm is unrealizable. It assumes perfect knowledge of the future. We would like to get some insight into $P_0(\omega,m)$ which may help us define good realizable algorithms. Unfortunately, we have not been able to come up with simplistic statements about when a page should or should not be kept in the memory since $P_0(\omega,m)$ is dependent upon $P_c(S,\omega,m)$, and $P_c(S,\omega,m)$ requires a dynamic programming solution. Therefore, the first part of this section presents results on the characteristics of the function $P_c(S,\omega,m)$ and the relative magnitudes of $P_c(S-\{p\},\omega,m)$ and $P_c(S,\omega,m)$. The second part of this section has several examples to illustrate the "gray area" in which dynamic programming may have to be used to determine the optimal strategy.

The first theorem concerns the size of the memory space that we allow a single program to occupy.

Theorem 3: For any $m \geq 1$, $P_c(S,\omega,m) \geq P_c(S,\omega,m+1)$.

This theorem states that by increasing the number of page frames that a single program may occupy, the total cost for processing the reference string may be decreased. This agrees with the well-known maxim "there is no substitute for real memory."

Proof: The proof consists of showing that any sequence of memory states which satisfies $P_c(S,\omega,m)$ also fits in a memory space of size m+1. Since $P_c(S,\omega,m+1)$ is the minimal cost when using a memory space of size m+1, $P_c(S,\omega,m) \geq P_c(S,\omega,m+1)$.

Q.E.D.

Before preceeding further, we will first present most of the definitions that are used throughout the remainder of this section. In the theorems that follow these definitions, we always assume that we are not bound by memory constraints.

Definition: The forward reference distance at time t for a page p and a reference string ω, denoted as $d_t(p,\omega)$ or simply as $d_t(p)$, is the distance in virtual time to the first reference to page p after time t, or

$$\begin{cases} k & \text{if } r_{t+k}=p \text{ and p does not appear} \end{cases}$$

$$d_t(p) = \begin{cases} \text{in the string } r_{t+1}r_{t+2}\cdots r_{t+k-1} \\ \infty \text{ if } p \text{ does not appear in the string} \\ r_{t+1}r_{t+2}\cdots r_T. \end{cases} \qquad (5.1)$$

Definition: Let $p \in S$ and $d_0(p) \neq \infty$. $P^-(S,\omega,m)$ is the cost of processing ω in a memory of size m with an initial memory state S according to the following algorithm A^-:

(i) record the optimal decisions for when starting with an initial memory state of $S-\{p\}$;

(ii) use the decisions recorded in (i), but start with an initial memory state of S.

Stated formally, the $\{Y_t\}$ decisions for A^- are given by $\{Y_t^-\}$.

$$Y_t^- = \begin{cases} \phi & \text{if } P_c(S_{t-1}-\{r_{t-1}\},\omega_t,m) \geq P_c(S_{t-1},\omega_t,m) \\ \{r_{t-1}\} & \text{if } P_c(S_{t-1}-\{r_{t-1}\},\omega_t,m) < P_c(S_{t-1},\omega_t,m) \end{cases} \qquad (5.2)$$

$$S_0 = S - \{p\}$$

$$S_t = \begin{cases} S_{t-1} - Y_t^- & \text{if } r_t \in S_{t-1} \\ S_{t-1} - Y_t^- + \{r_t\} & \text{if } r_t \notin S_{t-1} \end{cases}$$

Note that $\{S_t\}$ is not the sequence of memory states of A^-. These memory states may be described as

$$\begin{aligned} S_t^- &= S_t + \{p\} \qquad 0 \leq t < d_0(p) \\ S_t^- &= S_t \qquad\qquad d_0(p) \leq t \end{aligned} \qquad (5.3)$$

Thus, the only difference between algorithm A^- and the

decisions recorded involves keeping page p in main memory for the references $r_1 r_2 \cdots r_{d_0(p)-1}$ and a page fault at time $t=d_0(p)$. A similar cost may be defined using the optimal decisions when starting with an initial memory state of S.

Definition: Let $p \in S$ and $d_0(p) \neq \infty$. $P^+(S-\{p\},\omega,m)$ is the cost of processing ω in a memory of size m with an initial memory state S-{p} according to the following algorithm A^+:

(i) record the optimal decisions for when starting with an initial memory state of S;

(ii) use the decisions recorded in (i), but start with an initial memory state of S-{p}.

Stated formally, the $\{Y_t\}$ decisions for A^+ are given by $\{Y_t^+\}$.

$$Y_t^+ = \begin{cases} \emptyset & \text{if } P_c(S_{t-1}-\{r_{t-1}\},\omega_t,m) \geq P_c(S_{t-1},\omega_t,m) \\ \{r_{t-1}\} & \text{if } P_c(S_{t-1}-\{r_{t-1}\},\omega_t,m) < P_c(S_{t-1},\omega_t,m) \end{cases} \tag{5.4}$$

$$S_0 = S$$

$$S_t = \begin{cases} S_{t-1} - Y_t^+ & \text{if } r_t \in S_{t-1} \\ S_{t-1} - Y_t^+ + \{r_t\} & \text{if } r_t \notin S_{t-1} \end{cases}$$

At various times, it is necessary to refer to the sequences $\{S_t\}$, $\{X_t\}$, and $\{Y_t\}$ which result from following algorithm $P_0(\omega,m)$.

Definition: Let $\{S_t^0\}$, $\{X_t^0\}$, and $\{S_t^0\}$ represent the sequences $\{S_t\}$, $\{X_t\}$, and $\{Y_t\}$ respectively which result from applying algorithm $P_0(\omega,m)$ to the reference string ω.

Note that $P_c(S-\{p\},\omega,m) \geq P_c(S,\omega,m)$ implies $\{X_t^0\}=\{X_t^+\}$ (except for $t=d_0(p)$) and $\{Y_t^0\}=\{Y_t^+\}$. Also $P_c(S-\{p\},\omega,m)<P_c(S,\omega,m)$ implies $\{X_t^0\}=\{X_t^-\}$ (except for $t=d_0(p)$) and $\{Y_t^0\}=\{Y_t^-\}$ (for $t>1$).

With this background, we may now establish bounds on the difference $P_c(S_{t-1}-\{r_{t-1}\},\omega_t,m)-P_c(S_{t-1},\omega_t,m)$. It is easy to translate the meaning of these bounds into the forward reference distance of the page r_t and to decide whether it is cheaper to keep the page r_t in the memory or to remove it. Theorem 4 states these conditions.

Theorem 4: If $d_t(r_t) \leq \frac{r+(|S_t|+1)\cdot\Delta+\mu}{\Delta+\mu}$, then the optimal decision is to keep page r_t in the main memory until the next reference to it. Also, if $d_t(r_t) > \frac{r+|S_t|\cdot\Delta+\mu}{\mu}$, then the optimal decision is to remove page r_t from the main memory until the next reference to it, if any.

In order to prove Theorem 4, it is first necessary to prove the following lemma. This lemma will also be used extensively in the proofs of other theorems.

Lemma 4.1: If $p \in S$, $d_0(p) \neq \infty$ and $|S_t| < m$ $(1 \leq t < d_0(p))$, then

$$\tau + (|S| - \sum_{t=2}^{d_0(p)} |Y_t^-|) \cdot \Delta - (d_0(p)-1) \cdot \mu \leq P_c(S-\{p\},\omega,m) - P_c(S,\omega,m) \leq \tau + (|S| - \sum_{t=2}^{d_0(p)} |Y_t^+|) \cdot \Delta - (d_0(p)-1) \cdot \mu \qquad (5.5)$$

This lemma extablishes bounds on the difference $P_c(S-\{p\},\omega,m) - P_c(S,\omega,m)$ based on the sequences $\{Y_t^-\}$ and $\{Y_t^+\}$. The condition $|S_t| < m$ $(1 \leq t < d_0(p))$ is used in the proof to assure space in memory for page p during the references $r_1 r_2 \cdots r_{d_0(p)-1}$.

Proof: The first half of the lemma can be proven as follows:

$$|S^-_{d_0(p)}| = |S| + \sum_{t=1}^{d_0(p)} |X_t^-| - \sum_{t=1}^{d_0(p)} |Y_t^-| - 1, \text{ and } Y_1^- = \phi.$$

By the definitions of P_c and P^-,

$$P_c(S-\{p\},\omega,m) - P_c(S,\omega,m) \geq P_c(S-\{p\},\omega,m) - P^-(S,\omega,m). \qquad (5.6)$$

These differ only by the presence of page p for time $1 \leq t < d_0(p)$. Hence, we can rewrite (5.6) as

$$\begin{aligned} &P_c(S-\{p\},\omega,m) - P^-(S,\omega,m) \\ &= (d_0(p)-1) \cdot \rho + \sum_{t=1}^{d_0(p)-1} |S_t^-| \cdot \mu + \sum_{t=1}^{d_0(p)} |X_t^-| \cdot (\tau + |S_t^-| \cdot \Delta) + P_c(S^-_{d_0(p)}, \omega_{d_0(p)}, m) \\ &\quad - (d_0(p)-1) \cdot \rho - \sum_{t=1}^{d_0(p)-1} (|S_t^-|+1) \cdot \mu - \sum_{t=1}^{d_0(p)} |X_t^-| \cdot [\tau + (|S_t^-|+1) \cdot \Delta] \end{aligned}$$

$$-P_c(S^-_{d_0(p)},\omega_{d_0(p)},m)$$

$$= \tau+|S^-_{d_0(p)}|\cdot\mu-\sum_{t=1}^{d_o(p)-1}|X_t^-|\cdot\Delta-(d_0(p)-1)\cdot\mu$$

$$= \tau+(|S|+\sum_{t=1}^{d_o(p)}|X_t^-|-\sum_{t=1}^{d_o(p)}|Y_t^-|-1-\sum_{t=1}^{d_o(p)-1}|X_t^-|)\cdot\Delta-(d_0(p)-1)\cdot\mu$$

$$= \tau+(|S|-\sum_{t=1}^{d_o(p)}|Y_t^-|)\cdot\Delta-(d_0(p)-1)\cdot\mu$$

A similar argument may be made for the second half of the lemma using $P^+(S,\omega,m)$ and $|S^+_{d_0(p)}|=|S|+\sum_{t=1}^{d_o(p)}|X_t^+|-\sum_{t=1}^{d_o(p)}|Y_t^+|$.

Q.E.D.

<u>Proof of Theorem 4</u>: If $d_t(r_t)\leqslant\frac{\tau+(|S_t|+1)\cdot\Delta+\mu}{\Delta+\mu}$, then using Lemma 4.1,

$$P_c(S_t-\{r_t\},\omega_{t+1},m)-P_c(S_t,\omega_{t+1},m)$$

$$\geqslant \tau+(|S_t|-\sum_{k=1}^{d_t(r_t)}|Y_{t+k}^-|)\cdot\Delta-(d_t(r_t)-1)\cdot\mu$$

$$\geqslant \tau+(|S_t|-d_t(r_t)+1)\cdot\Delta-(d_t(r_t)-1)\cdot\mu$$

$$\geqslant 0$$

or $P_c(S_t-\{r_t\},\omega_{t+1},m)\geqslant P_c(S_t,\omega_{t+1},m)$.

If $d_t(r_t)=\infty$, then it is true that $P_c(S_t-\{r_t\},\omega_{t+1},m)<P_c(S_t,\omega_{t+1},m)$. If $d_t(r_t)>\frac{\tau+|S_t|\cdot\Delta+\mu}{\mu}$ and $d_t(r_t)\neq\infty$, then we may use Lemma 4.1.

$$P_c(S_t-\{r_t\},\omega_{t+1},m)-P_c(S_t,\omega_{t+1},m)$$

$$\leqslant \tau+(|S_t|-\sum_{k=1}^{d_t(r_t)}|Y_{t+k}^+|)\cdot\Delta-(d_t(r_t)-1)\cdot\mu$$

$$< \tau + |S_t| \cdot \Delta - (d_t(r_t)-1) \cdot \mu$$
$$< 0$$

or $P_c(S_t - \{r_t\}, \omega_{t+1}, m) < P_c(S_t, \omega_{t+1}, m)$.

Q.E.D.

Using the result of Theorem 4, we prove that the algorithm of Prieve and Fabry [P1] is optimal for the cost function assumed by them. This is a special case of the general problem with $\Delta=0$.

Corollary 4.1: If Δ is zero, then $P_0(\omega,m)$ may be restated as follows:

$$S_0 = \emptyset$$

$$S_{t+1} = \begin{cases} S_t & \text{if } r_{t+1} \in S_t \wedge d_t(r_t) \leqslant \frac{\tau+\mu}{\mu} \\ S_t - \{r_t\} & \text{if } r_{t+1} \in S_t \wedge d_t(r_t) > \frac{\tau+\mu}{\mu} \\ S_t + \{r_{t+1}\} & \text{if } r_{t+1} \notin S_t \wedge d_t(r_t) \leqslant \frac{\tau+\mu}{\mu} \\ S_t - \{r_t\} + \{r_{t+1}\} & \text{if } r_{t+1} \notin S_t \wedge d_t(r_t) < \frac{\tau+\mu}{\mu} \end{cases} \qquad (5.7)$$

The optimal decision is to remove r_t if $d_t(r_t) > \frac{\tau+\mu}{\mu}$.

Proof: It is only necessary to prove that $d_t(r_t) \leqslant \frac{\tau+\mu}{\mu}$ implies the optimal decision is to keep page r_t in the main memory, and $d_t(r_t) > \frac{\tau+\mu}{\mu}$ implies the optimal decision is to remove page r_t from the main memory. But with $\Delta=0$, this is so.

Q.E.D.

One would intuitively expect that if the optimal decision is to keep a page p_1 in memory between two references and a page p_2 is referenced twice between the references to p_1, then the optimal decision should be to keep p_2 in memory between references. The reference string may be pictured as

$$---p_1--p_2-----p_2-p_1--$$

This is indeed the case as shown by Theorem 5.

Theorem 5: If $p \in S$, $P_c(S-\{p\},\omega,m) \geq P_c(S,\omega,m)$ and there exists a t such that $d_t(r_t)+t<d_0(p)$, then $P_c(S_t^0-\{r_t\},\omega_{t+1},m) \geq P_c(S_t^0,\omega_{t+1},m)$.

Proof: Assume the contrary or that $P_c(S_t^0-\{r_t\},\omega_{t+1},m)<P_c(S_t^0,\omega_{t+1},m)$. This means that the optimal decision is to remove page r_t from memory at time t+1. From the proof of Lemma 4.1, we have

$$\begin{aligned} 0 &> P_c(S_t^0-\{r_t\},\omega_{t+1},m)-P_c(S_t^0,\omega_{t+1},m) \\ &\geq \tau+(|S_t^0|-\sum_{k=1}^{d_t(r_t)}|Y_{t+k}^-|)\cdot\Delta-(d_t(r_t)-1)\cdot\mu \\ &= \tau+(|S_t^0|-\sum_{k=1}^{d_t(r_t)}|Y_{t+k}^0|)\cdot\Delta-(d_t(r_t)-1)\cdot\mu \end{aligned} \qquad (5.8)$$

We also have,

$$\begin{aligned} 0 &\leq P_c(S-\{p\},\omega,m)-P_c(S,\omega,m) \\ &\leq \tau+(|S|-\sum_{k=1}^{d_0(p)}|Y_k^+|)\cdot\Delta-(d_0(p)-1)\cdot\mu \end{aligned}$$

$$= \tau+(|S|-\sum_{k=1}^{d_o(p)}|Y_k^0|)\cdot\Delta-(d_0(p)-1)\cdot\mu \qquad (5.9)$$

Finally, $|S_t^0|=|S|+\sum_{k=1}^{t}|X_k^0|-\sum_{k=1}^{t}|Y_k^0|$. Thus, we have

$$0 > \tau+(|S_t^0|-\sum_{k=1}^{d_t(r_t)}|Y_{t+k}^0|)\cdot\Delta-(d_t(r_t)-1)\cdot\mu$$

$$= \tau+(|S|+\sum_{k=1}^{t}|X_k^0|-\sum_{k=1}^{t}|Y_k^0|-\sum_{k=1}^{d_t(r_t)}|Y_{t+k}^0|)\cdot\Delta-(d_t(r_t)-1)\cdot\mu$$

$$> \tau+(|S|-\sum_{k=1}^{t+d_t(r_t)}|Y_k^0|)\cdot\Delta-(d_t(r_t)-1)\cdot\mu$$

$$> \tau+(|S|-\sum_{k=1}^{d_o(p)}|Y_k^0|)\cdot\Delta-(d_t(r_t)-1)\cdot\mu$$

$$> \tau+(|S|-\sum_{k=1}^{d_o(p)}|Y_k^0|)\cdot\Delta-(d_0(p)-1)\cdot\mu$$

$$> 0$$

Since this is a contradiction, $P_c(S_t^0-\{r_t\},\omega_{t+1},m)\geq P_c(S_t^0,\omega_{t+1},m)$.

Q.E.D.

This theorem characterizes a relationship between the optimal policy and the set of distinct pages referenced between two references to the same page. The following definition and corollaries explain the relationships.

Definition: Let $R_t=\{p \mid p=r_{t+i}\ 1\leq i<d_t(r_t)\}$ be the set of distinct pages referenced at least once between the reference to r_t and the next reference to r_t. (Sometimes, in the literature, $|R_t|$ is referred to as the forward LRU stack distance.)

Corollary 5.1: If $P_c(S_t-\{r_t\},\omega_{t+1},m)\geq P_c(S_t,\omega_{t+1},m)$, then $\sum_{k=1}^{d_t(r_t)}|Y^0_{t+k}|\leq|R_t|$.

If the optimal policy is to keep r_t in memory between two references, then the number of pages removed during the same interval is less than or equal to the number of distinct pages referenced.

Proof: Assume that $\sum_{k=1}^{d_t(r_t)}|Y^0_{t+k}|>|R_t|$. Then there exists a page p such that $Y^0_i=Y^0_j=\{p\}$ and $t+1<i<j\leq t+d_t(r_t)$. Then $r_{i-1}=r_{j-1}=p$, or p was referenced twice in the interval and $P_c(S_{i-1}-\{r_{i-1}\},\omega_i,m)<P_c(S_{i-1},\omega_i,m)$. This is in contradiction to Theorem 5 and thus the corollary is valid.

Q.E.D.

Corollary 5.2: If $P_c(S_t-\{r_t\},\omega_{t+1},m)<P_c(S_t,\omega_{t+1},m)$, then $S_t\subseteq R_t$.

If the optimal policy is to remove page r_t from memory, then the memory state must be a subset of the pages referenced.

Proof: If $S_t\not\subseteq R_t$, then there exists a $r_{t'}\in S_t$ such that $P_c(S_{t'}-\{r_{t'}\},\omega_{t'+1},m)\geq P_c(S_{t'},\omega_{t'+1},m)$ and $t'+d_{t'}(r_{t'})>t+d_t(r_t)$. By Theorem 5,

$P_c(S_t-\{r_t\},\omega_{t+1},m) \geq P_c(S_t,\omega_{t+1},m)$, which is a contradiction.

Q.E.D.

Now we prove a new lower bound on $d_t(r_t)$ which is superior to the bound in Theorem 4 in some cases.

Theorem 6: If the optimal decision is to remove page r_t at time t+1, then $d_t(r_t) > \frac{\tau+(|S_t|-|R_t|)\cdot\Delta+\mu}{\mu}$.

Proof: The proof of this theorem will consist of the following lemmas. Each lemma will prove a special case of Theorem 6, but when taken together, they are the equivalent of Theorem 6.

Lemma 6.1: If $P_c(S_t-\{r_t\},\omega_{t+1},m) < P_c(S_t,\omega_{t+1},m)$, and $\sum_{k=2}^{d_t(r_t)} |Y^-_{t+k}| = \sum_{k=2}^{d_t(r_t)} |Y^0_{t+k}| \leq |R_t|$, then $d_t(r_t) > \frac{\tau+(|S_t|-|R_t|)\cdot\Delta+\mu}{\mu}$.

Proof: By Lemma 4.1,

$$0 > P_c(S_t-\{r_t\},\omega_{t+1},m) - P_c(S_t,\omega_{t+1},m)$$

$$\geq \tau+(|S_t|-\sum_{k=2}^{d_t(r_t)}|Y^-_{t+k}|)\cdot\Delta-(d_t(r_t)-1)\cdot\mu$$

$$\geq \tau+(|S_t|-|R_t|)\cdot\Delta-(d_t(r_t)-1)\cdot\mu$$

or $d_t(r_t) > \frac{\tau+(|S_t|-|R_t|)\cdot\Delta+\mu}{\mu}$.

Q.E.D.

The following lemmas have a condition $\sum_{k=2}^{d_t(r_t)} |Y^0_{t+k}| > |R_t|$. This condition can be paraphrased to mean that the optimal strategy requires more than $|R_t|$ pages to be removed from main memory between the two references to r_t. However, since there are only $|R_t|$ distinct pages referenced during that interval, at least one page must have been removed from memory at least twice. We will define $p+d_p(r_p)$ to be the first occurrence after r_t where a page, r_p, was referenced after being removed from memory after time t. This may be stated more formally as $p>t$, $P_c(S_p-\{r_p\},\omega_{p+1},m)<P_c(S_p,\omega_{p+1},m)$ and for any $t'>t$ with $P_c(S_{t'}-\{r_{t'}\},\omega_{t'+1},m)<P_c(S_{t'},\omega_{t'+1},m)$, $p+d_p(r_p)<t'+d_{t'}(r_{t'})$. Finally, $p+d_p(r_p)<t+d_t(r_t)$, $d_p(r_p)<d_t(r_t)$, and $|R_p|<|R_t|$.

Lemma 6.2: If p is defined as described above, then $\sum_{k=2}^{d_p(r_p)} |Y^0_{p+k}| \leqslant |R_p|$.

Proof: If $\sum_{k=2}^{d_p(r_p)} |Y^0_{p+k}| > |R_p|$, then there must be a page that is referenced twice and removed from the memory between the references to page p. If we call this page $r_{t'}$, then $t'+d_{t'}(r_{t'})<p+d_p(r_p)$ and $P_c(S_{t'}-\{r_{t'}\},\omega_{t'+1},m)<P_c(S_{t'},\omega_{t'+1},m)$. This is in contradiction to the definition of p, and thus $\sum_{k=2}^{d_p(r_p)} |Y^0_{p+k}| \leqslant |R_p|$.

Q.E.D.

Lemma 6.3: If $P_c(S_t-\{r_t\},\omega_{t+1},m)<P_c(S_t,\omega_{t+1},m)$, $\sum_{k=2}^{d_t(r_t)}|Y^0_{t+k}|>|R_t|$, p is as defined above, and $|S_p|\geq|S_t|$, then $d_t(r_t)>\frac{\tau+(|S_t|-|R_t|)\cdot\Delta+\mu}{\mu}$.

Proof: By Lemma 4.1,

$$0 > P_c(S_p-\{r_p\},\omega_{p+1},m)-P_c(S_p,\omega_{p+1},m).$$

$$\geq \tau+(|S_p|-\sum_{k=2}^{d_p(r_p)}|Y^0_{p+k}|)\cdot\Delta-(d_p(r_p)-1)\cdot\mu$$

$$\geq \tau+(|S_p|-|R_p|)\cdot\Delta-(d_p(r_p)-1)\cdot\mu$$

$$> \tau+(|S_t|-|R_t|)\cdot\Delta-(d_p(r_p)-1)\cdot\mu$$

$$> \tau+(|S_t|-|R_t|)\cdot\Delta-(d_t(r_t)-1)\cdot\mu$$

or $d_t(r_t)>\frac{\tau+(|S_t|-|R_t|)\cdot\Delta+\mu}{\mu}$.

Q.E.D.

Lemma 6.4: If p is defined as described above and $|S_p|<|S_t|$ or $n=|S_t|-|S_p|$, then $|R_p|<|R_t|-n$.

Proof: Define $Y=\bigcup_{k=1}^{p}Y^0_{t+k}$ and $P=\{r_{t+k} \mid 1\leq k<p+d_p(r_p)\}$. Since $|S_p|=|S_t|+\sum_{k=1}^{p}|X^0_{t+k}|-\sum_{k=1}^{p}|Y^0_{t+k}|=|S_t|+\sum_{k=1}^{p}|X^0_{t+k}|-|Y|$, $|Y|\geq n$. (Note that $\sum_{k=1}^{p}|Y^0_{t+k}|=|Y|$; if this were not true, then there would be a contradiction to the definition of p.) Also, $|P|\leq|R_t|$ and $|P|\geq|R_p\cup Y|$.

Also, $R_p \cap Y = \emptyset$ since, by the definition of p, p is the first page removed from memory and then referenced again. Therefore, $|P| \geq |R_p \cup Y| = |R_p| + |Y| \geq |R_p| + n$, or $|R_p| \leq |P| - n \leq |R_t| - n$.

Q.E.D.

Lemma 6.5: If $P_c(S_t - \{r_t\}, \omega_{t+1}, m) < P_c(S_t, \omega_{t+1}, m)$, $\sum_{k=2}^{d_t(r_t)} |Y^0_{t+k}| > |R_t|$, p is defined as above, and $|S_p| < |S_t|$, then $d_t(r_t) > \frac{\tau + (|S_t| - |R_t|) \cdot \Delta + \mu}{\mu}$.

Proof: Let $n = |S_t| - |S_p|$, then $|R_p| \leq |R_t| - n$. Using Theorem 3,

$$
\begin{aligned}
0 &> P_c(S_p - \{r_p\}, \omega_{p+1}, m) - P_c(S_p, \omega_{p+1}, m) \\
&\geq \tau + (|S_p| - \sum_{k=2}^{d_o(p)} |Y^0_{p+k}|) \cdot \Delta - (d_p(r_p) - 1) \cdot \mu \\
&\geq \tau + (|S_p| - |R_p|) \cdot \Delta - (d_p(r_p) - 1) \cdot \mu \\
&\geq \tau + (|S_t| - n - |R_t| + n) \cdot \Delta - (d_p(r_p) - 1) \cdot \mu \\
&> \tau + (|S_t| - |R_t|) \cdot \Delta - (d_t(r_t) - 1) \cdot \mu
\end{aligned}
$$

or $d_t(r_t) > \frac{\tau + (|S_t| - |R_t|) \cdot \Delta + \mu}{\mu}$.

Q.E.D.

The above series of lemmas proves Theorem 6.

Q.E.D.

Theorem 6 can be restated in a more useable form.

Corollary 6.1: If $d_t(r_t) \leq \frac{\tau+(|S_t|-|R_t|)\cdot A+\mu}{\mu}$ then the optimal decision is to keep page r_t in the main memory until the next reference to it.

Proof: This is the contrapositive of Theorem 6.

Q.E.D.

The major results are summarized in Table 1. However, these results still leave a "gray area" where dynamic programming is required to compute the difference $P_c(S_t-\{r_t\},\omega_{t+1},m)-P_c(S_t,\omega_{t+1},m)$. This "gray area" is examined through the use of several examples.

The first example in figure 1 demonstrates that the bound derived in Corollary 6.1 may be the least upper bound for making the decision to keep $\{r_t\}$ in memory without using dynamic programming. The second example in figure 1 demonstrates that the condition is not both sufficient and necessary, but merely sufficient.

Figure 2 contains an example which demonstrates that the bound derived in Theorem 4 may be the greatest lower bound for making the decision to remove a page from memory without using dynamic programming.

The examples above have shown that we may not eliminate dynamic programming just by comparing the forward reference distance to an easily computable number. However, it might

be possible to derive good results from a comparison of a reference string to a "standard" reference string. The next two examples question the usefulness of a set of "standard" reference strings.

The first question concerns the set of pages referenced between two references to the same page. In particular, if $d_t(r_t,\omega)=d_t(r_t,\omega')$, $\omega=r_1r_2 \cdots r_tr_{t+1} \cdots r_T$, $\omega'=r_1r_2 \cdots r_tr'_{t+1} \cdots r'_T$, and $R_t(\omega)=R_t(\omega')$, is the optimal decision similar for both reference strings at time t? The answer is shown to be no in figure 3. To even closely approximate the optimal strategy would require such a large set of "standard" reference strings as to be even more impractical than dynamic programming.

Another question centers upon common subsequences. In figure 4, the common subsequence is 1 2 3 4 3 2 1. In addition, the number of pages in the memory state at the beginning of the subsequence is the same for both reference strings. However, the results are quite different, and thus the context of a subsequence is important.

In summary, this section has explored various properties of the function $P_c(S_t,\omega_{t+1},m)$, and determined some cases where dynamic programming is not required to achieve an answer for the relative magnitudes of $P_c(S_t-\{t_t\},\omega_{t+1},m)$ and $P_c(S_t,\omega_{t+1},m)$. However, we have also

given some examples where the relative magnitudes can not be determined by our results without using dynamic programming.

Conclusions

We have developed a cost function expressed in terms of the memory used and the page faults occurring during the processing of a reference string. This cost function is more realistic than the cost functions that have been used so far in the literature. It is also an approximation of the space-time product criteria. An algorithm $P_0(\omega,m)$ has been proposed in section 4 which is optimal for the cost function. Since $P_0(\omega,m)$ is unrealizable, its main use is as a theoretical lower bound that can be achieved by any variable partition algorithm. $P_0(\omega,m)$ essentially decides after each reference whether it is cheaper to keep the page most recently referenced in the memory until the next reference or to throw it out and recall it on demand. Since no specific properties of the reference strings are assumed this decision is in general based on the reference pattern of all the pages in the memory. However, if the forward distance is larger than the bound in Theorem 4 or lower than the bound in Theorem 4 or Corollary 6.1, the optimal decision is easy to take. Currently work is under way to make these bounds tighter for programs which can be

characterized by some specific type of reference string. It may be possible to extend the analysis using the cost function proposed here to those types of program behavior models where the probabilistic behavior of a page is dependent upon the other pages present in the memory.

Condition	Implication
$d_t(r_t) \leqslant \frac{\tau+(\lvert S_t\rvert+1)\cdot\Delta+\mu}{\Delta+\mu}$	keep r_t in memory
$d_t(r_t) \leqslant \frac{\tau+(\lvert S_t\rvert-\lvert R_t\rvert)\cdot\Delta+\mu}{\mu}$	keep r_t in memory
$d_t(r_t) > \frac{\tau+\lvert S_t\rvert\cdot\Delta+\mu}{\mu}$	remove r_t from memory

Table 1. Major Results

$$\omega = 1\ 2\ 3\ 4\ 5\ 1$$

$$d_1(1)=5 \qquad S_1=\{1\} \qquad R_1=\{2,3,4,5\}$$

$$\tau=7-\epsilon \qquad \Delta=1 \qquad \mu=1 \qquad m\geqslant 2$$

$$d_1(1)=5 > \frac{\tau+(\lvert S_1\rvert-\lvert R_1\rvert)\cdot\Delta+\mu}{\mu}=5-\epsilon$$

$$P_c(\phi,\omega_2,m)<P_c(\{1\},\omega_2,m)$$

$$\omega = 1\ 2\ 3\ 1\ 2\ 3$$

$$d_1(1)=3 \qquad S_1=\{1\} \qquad R_1=\{2,3\}$$

$$\tau=2 \qquad \Delta=2 \qquad \mu=1 \qquad m\geqslant 3$$

$$d_1(1)=3 > \frac{\tau+(\lvert S_1\rvert-\lvert R_1\rvert)\cdot\Delta+\mu}{\mu}=1$$

$$P_c(\phi,\omega_2,m)>P_c(\{1\},\omega_2,m)$$

Figure 1.

$$\omega = 2\ 1\ 2\ 2\ 2\ 2\ 2\ 1\ 2$$

$d_2(1)=6 \qquad S_2=\{1,2\} \qquad R_2=\{2\}$

$\tau=3 \qquad \Delta=1 \qquad \mu=1 \qquad m\geq 2$

$$d_2(1)=6 < \frac{\tau+|S_2|\cdot\Delta+\mu}{\mu}=6$$

$$P_c(\{2\},\omega_3,m)\geq P_c(\{1,2\},\omega_3,m)$$

Figure 2.

$$\omega = 1\ 2\ 3\ 4\ 4\ 4\ 2\ 3$$

$$\omega' = 1\ 2\ 3\ 4\ 2\ 4\ 4\ 3$$

$$D_3(3,\omega) = 5 = d_3(3,\omega')$$

$$R_3(\omega) = \{2,4\} = R_3(\omega')$$

$\tau>4\mu \qquad \Delta>\tau-4\mu$

$S_3(\omega)=\{3\} \qquad S_3(\omega')=\{2,3\}$

$Y_3^0(\omega)=\{3\} \qquad Y_3^0(\omega')=\emptyset$

Figure 3.

$\tau=7.5\mu \qquad \Delta=5.0\mu \qquad m>3$

ω =	3	1	2	3	4	3	2	1
$\{S_t^0\}$ =	{3}	{1,3}	{2,3}	{2,3}	{2,3,4}	{2,3}	{2}	{1}
ω' =	2	1	2	3	4	3	2	1
$\{S_t^0\}$ =	{2}	{1,2}	{2}	{3}	{3,4}	{3}	{2}	{1}

Figure 4.

References

[A1] Aho, A.V., P.J. Denning, and J.D. Ullman, "Principles of Optimal Page Replacement," J. of the ACM, Vol. 18, No. 1, January 1971, pp. 80-93.

[A2] Arvind, R.Y. Kain, and E. Sadeh, "On Reference String Generation Processes," Fourth Symposium on Operating Systems Principles, Operating Systems Review, Vol. 7, No. 4, October 1973, pp. 80-87.

[B1] Belady, L.A., "A Study of Replacement Algorithms for Virtual Storage Computers," IBM Systems J., Vol. 5, No. 2, 1966, pp. 78-101.

[B2] Belady, L.A. and F.P. Palermo, "On-line Measurement of Paging Behavior by the Multivalued MIN Algorithm," IBM J. of Research and Development, Vol. 18, No. 1, January 1974, pp. 2-19.

[B3] Bellman, R., Dynamic Programming, Princeton University Press, Princeton, New Jersey, 1957.

[C1] Coffman, E.G. and P.J. Denning, Operating Systems Theory, Prentice-Hall, Englewood Cliffs, New Jersey, 1973, Chapters 6 and 7.

[C2] Coffman, E.G., Jr. and T.A. Ryan, Jr., "A Study of Storage Partitioning Using a Mathematical Model of Locality," Communications of the ACM, Vol. 15, No. 3, March 1972, pp. 185-190.

[D1] Denning, P.J., "The Working Set Model for Program Behavior," Communications of the ACM, Vol. 11, No. 5, May 1968, pp. 323-333.

[D2] Denning, P.J., "Virtual Memory," Computing Surveys, Vol. 2, No. 3, September 1970, pp. 153-189.

[D3] Denning,P.J., J.E. Savage, and J.R. Spirn, "Models for Locality in Program Behavior," Technical Report, Princeton University, Princeton, New Jersey, April 1972.

[K1] King, W.F.,III, "Analysis of Paging Algorithms," IFIP Conference Proceedings, Ljubljana, Yugoslavia, August 1971.

[M1] Mattson, R.L., J. Gecsei, D.R. Slutz, and I.W. Traiger, "Evaluation Techniques for Storage Hierarchies," IBM Systems J., Vol. 9, No. 2, 1970, pp. 78-117.

[P1] Prieve, B.G. and R.S. Fabry, "An Optimal Variable Space Page Replacement Algorithm," Technical Report, Bell Laboratories, Naperville, Illinois, May 1974.

[S1] Shedler, G.S. and C. Tung, "Locality in Page Reference Strings," SIAM Journal on Computing, Vol. 1, No. 3, September 1972, pp. 218-241.